Stream-IT: Continuous and dynamic processing of production systems data - Throughput bottlenecks as a case-study

Stream-IT: Continuous and dynamic processing of production systems data - throughput bottlenecks as a case-study

Hannaneh Najdataei, Mukund Subramaniyan, Vincenzo Gulisano, Anders Skoogh, Marina Papatriantafilou
Chalmers University of Technology, Sweden
{hannajd,mukunds,vincenzo.gulisano,anders.skoogh,ptrianta}@chalmers.se

***Abstract*—Considering the needs for continuous availability of information out of data generated in Cyber-Physical production systems, we investigate the use of continuous stream processing as a paradigm for generating useful information out of the data, to support efficient and safe operation, as well as planning activities.**

Our contributions and expected benefits: (i) we show possibilities to automate and pipeline the validation and analysis of the data, hence providing an automated way to improve the quality of the latter and parallelizing the two phases; (ii) we show how to induce lower latency in generating the desired information, enabling it to be continuously made available, before whole batches of data are gathered, in cost-efficient ways; (iii) besides the automation of the above procedures that are commonly done in a batch fashion and with significant manual effort by the production system analysts, we show additional options for configuring ways in which to automate deeper analysis of the data; in particular, we provide evidences about how the rich semantics of stream processing frameworks can ease the development and deployment of data analysis applications in production systems.

Moreover, using the problem of bottleneck detection as a sample scenario, we illustrate the above in a concrete fashion, on cost-efficient systems, that are plausible to have in existing deployments. The experimental study is on a 2-year data-set with more than 8.5 million entries, from a system including more than 30 interconnected machines and it demonstrates the benefits of the proposed methods, in providing timely and multi-dimensional information from the data, enabling possibilities for deeper analyses.

I. Introduction

New and upcoming production systems, empowered with sensing and communication devices, generate a lot of data that can be used to improve the functionality and resource utilization of the system, as well as to react to risky situations. To this end, the log of the system's activities is provided using the machine information captured by Manufacturing Execution Systems (MES) [8]. MES provide massive amount of raw data whose translation into smaller sets of valuable information is challenging for the production system analyst. The data is often characterized as Big Data and as Doug Laney [9] (from Gartner, Inc) emphasized, "is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation".

An additional challenge is given by the fact that *data validation and quality improvement* [1] need to be applied on the data before performing different analysis. The data validation commonly involves: (i) machine data extraction from MES, (ii) data sanitization, and (iii) data organizing based on analysis parameters. Out of the three generic steps, the sanitization usually takes the longest time and includes operations such as removing information outside the time interval of interest, extreme outliers, and faulty signals. Identification of extreme outliers and faulty signals and decisions to retain or remove them before further analysis are highly subjective in nature and depend on the analyst's interpretation [3]. Therefore, significant manual efforts are required for the analyst to validate the data, which shows a demand of configurable automated tools for data validation.

Challenges: The noisy and dynamic data that is generated, often irregularly, from various sources requires constant validation before the analysis procedure. At the same time, in a production system, online MES data analysis is required for live information to be leveraged in scenarios such as failure detection. Currently, based on common methods in the literature and in practice, the raw data is initially stored until, later on, the system analyst triggers some analysis process. However, this implies that the information about the data that is processed most often comes with latency that is prohibitive for taking timely decisions.

Contributions: In this paper, we design and implement an analysis approach for production systems that addresses the above mentioned challenges relying on the *data stream processing* paradigm.

Data stream processing (aka data streaming) is an alternative to the traditional store-then-process paradigm. It defines methods by which data is processed continuously, as they are generated, so that the extracted information can enable to react to changing conditions in a continuous fashion. We

The authors would like to thank the FFI programme (funded by VINNOVA, the Swedish Energy Agency, and the Swedish Transport Administration) for their funding of the Data Analytics in Maintenance Planning research project (DAiMP) [Grant number: 2015-06887] and the Swedish Foundation for Strategic Research (SSF), for their funding of project FiC [Grant number: GMT14-0032], under which this research was conducted. The authors would also like to thank Anders Ramström, who furnished the real-time data from a real-world production system.

978-1-7281-3666-0/19/$31.00 ©2019 IEEE

argue that stream processing enables configurable analysis to provide options to the analyst to generate helpful information for automating deeper parts of the analysis. Another advantage of stream processing is that there is no need to store massive data (if that is e.g. not needed or is sensitive) but only the results of the validated data [6], [16], or even only the results of the analysis if the rest is not needed or can be recovered in critical scenarios through streaming data-provenance [12]. We provide the following contributions:

1) we propose an automated configurable streaming-based method for data validation to sanitize and organize MES data for further processes;
2) we show insights on continuous, low-latency data processing as a methodology in production systems;
3) we also study the proposed approach through an implementation of a streaming-based analysis on top of an established *Stream Processing Engine* (SPE), Apache Flink [4], on the problem of *throughput bottleneck detection*, automating and dramatically enhancing the efficiency of an established approach, i.e. the *active period method for bottleneck detection* [13], [15].

The experimental study is on a 2-year data-set with more than 8.5 million entries, from an automotive production system including more than 30 interconnected machines and it demonstrates the benefits of the proposed methods, in providing timely and multi-dimensional information from the data, enabling possibilities for deeper analyses.

II. Preliminaries

In this section, we describe key properties of data stream processing. We also provide detailed analysis description of throughput bottleneck detection as a case study.

A. Data Stream Processing

The data stream processing paradigm was motivated by continuous processing of flows of data targeting data-intensive applications for which the traditional store-then-process paradigm is not suitable because of the analysis latency or system memory limitations. In data streaming, a query is used constantly on flow of data on the fly as input to produce a continuous flow of data as output, therefore, causes less latency by continuously producing the results without waiting for the a batch of data to be available.

In addition to above, stream processing enables pipeline parallelism which can be exploited more by performing on multi-core systems. The development of data streaming paradigm, has lead to real-time processing and timeliness of the results in applications (e.g. sensor readings) where the two steps of capturing and processing data can be merged.

The flow of data, also called data stream, is a sequence of data records, which are referred as tuples, that share a common schema, for instance $< ts, A_1, A_2, ..., A_n >$ where ts is the creation timestamp and $A_1, ..., A_n$ are the attributes of the tuple. To deal with the tuples, SPEs are designed to provide high level programming interfaces. They use continuous queries in the form of Directed Acyclic Graphs (DAGs) where *vertices* represent stream processing *operators* to enforce queries and directed *edges* represent flow of data. An operator can be *stateless* or *stateful*. Stateless operators perform actions based on each tuple individually while stateful operators produce results using several tuples, commonly over a *window* of data which contain the most recent tuples. Windows are defined by two parameters *size*, the length of the window, and *advance*, the sliding amount (how much in time the window should go forward). In the following, we overview definitions of some of the basic streaming operators:

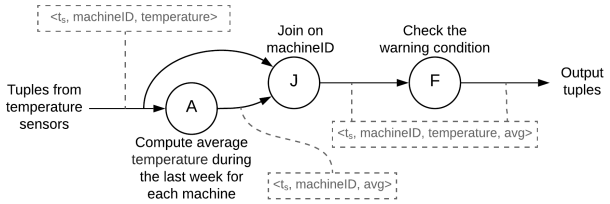

Fig. 1: Sample query for temperature monitoring example

- **Filter** is a stateless operator, for each tuple produces zero or one tuple by forwarding or discarding it based on a filtering condition.
- **Map** is a stateless operator, transforms each tuple from one schema to another and produces zero, one or more tuples based on the mapping condition.
- **Aggregate** is a stateful operator, used to aggregate information from multiple tuples over a window and produces one tuple per window.
- **Join** is a stateful operator, used to match receiving tuples from two streams using a given predicate function and produces one tuple per matching.

In the following, we discuss an example of using stream queries for a production system.

Temperature monitoring example: For each machine in the production line, report a warning if the temperature of the machine is two times higher than its average temperature over the last week. Figure 1 presents a sample query to solve this problem. As shown in the figure, the query receives tuples with timestamp, unique machine id, and the machine's temperature. An Aggregate operator is, then, performed on incoming tuples to compute the average temperature of each machine over a window of size one week and produces tuples with an attribute average temperature. Afterwards, a Join operator is used to match the original tuples with calculated average ones based on the machineID attribute. The matched tuple contains both attributes temperature and average temperature. Therefore, by checking the warning condition in the Filter operator, if the temperature is two times higher than the average, the tuple will be forwarded to output otherwise it will be discarded.

B. Throughput bottleneck detection

In industrial practice, it is often required from the analysts to monitor throughput of the production system, one of the

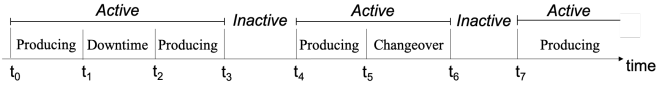

Fig. 2: Active and inactive machine states (adopted from [13])

main indicators of system performance [5]. Throughput is commonly affected by one or more machines in the production system and these machines are thus referred to as throughput bottlenecks. Bottlenecks need to be monitored on a real-time basis to reduce the response time to implement improvement actions and thereby facilitate real-time production control [18].

Substantial research efforts have been devoted to developing algorithms for the bottleneck detection problem over the last two decades. These algorithms make use of the machine information captured by the MES [8]. For instance, Hopp et al. [7] uses the machines utilization information to detect the bottlenecks. In other works [2], [10], [14], the blockage and starvation machine information are used to detect the bottlenecks. Subramaniyan et al. [15] use machines' active states information to detect the bottlenecks based on active period theory [13]. Yu et al. [17] improved the statistical analysis of the bottlenecks. These algorithms are demonstrated to detect the historical bottlenecks from the machine data collected over a period of time (e.g. past hours, past shifts, past days, etc.). In this paper, we study the throughput bottleneck detection based on the active period theory [13], [15], which, for self-containment, we paraphrase here from [13].

Active Period Method for Bottleneck Detection: The idea behind the active period theory of bottleneck detection is to measure the duration of machines being active. Therefore, the machine states need to be classified during a production run into those in active and inactive states, i.e. when the machine is waiting for a part or a service or waiting for the removal of a part, it is in an inactive state otherwise it is in an active state. Moreover, consecutive active states are considered as one with the duration equal to the sum of duration of the individual active states. Figure 2 indicates an example of active and inactive states of one machine. Let $A_m = \{a_{m1}, a_{m2}, ..., a_{mn}\}$ be a set of active duration, where $m \in 1, ..., M$ is the machine id and $n \in 1, ..., N$ in the index of the measured active duration. The active period percentage $\overline{a}_m$ is calculated as:

$$\overline{a}_m = \frac{\sum_{n=1}^{N} a_{mn}}{N}$$

The machine with the longest average active period is considered to be the bottleneck. In another words, the activity of the machine with the highest average active period is interrupted by the other machines the least, which in turn makes this machine to effect the overall system throughput the most.

III. Methodology

In this section we describe the proposed architecture which consists of two modules: data validation and bottleneck detection. As mentioned earlier, regardless of the processing application, analyzing MES data contains one major validating round. In order to support a configurable and automated analysis architecture, it is necessary to automate both data validating and processing rounds. To this end, we first provide a comparison between the streaming and conventional analysis for the studied use-case. Later, we present the generic automated and streaming-based validation module of the architecture which can be used on MES data before applying any analysis application. Finally, we describe the streaming-based bottleneck detection module.

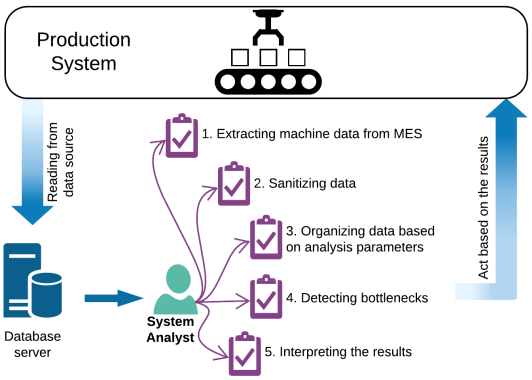

Fig. 3: Layout of the conventional bottleneck detection.

A. Conventional vs. Streaming Processing

Figure 3 overviews the conventional processing that is commonly used for active period based bottleneck detection algorithm. As shown in the figure, all tasks should be done sequentially and manually by the system analyst. This causes an unnecessary excessive latency to produce results. Another disadvantage of conventional processing is that the procedure is limited by the speed of storing the data since it is necessary to first store all data and then run the analysis. However, as discussed in Section II, due to the pipeline parallelism in the stream processing, it is possible to pipline storing the data with processing procedure. Furthermore, conventional processing is sensitive to the time period of interest for which bottlenecks need to be detected. Therefore, by increasing the time period, the analysis takes even longer which is undesirable.

Here we discuss the role of the concept of the time period (window) in stateful streaming data analysis in the context of interest. I.e. using stream processing we can produce results for different window sizes at the same time, hence revealing bottlenecks at different levels of time-granularity.

Dependencies among machines in a pipeline make dynamic throughput-bottleneck detection difficult and in need of dynamic, time-sensitive monitoring; e.g. if we think of machine pipelines as directed graphs, removing a bottleneck from an upstream machine can reveal another one, downstream. In this regard, performing the bottleneck detection procedure on different window sizes concurrently (from sec to hours, days, weeks, months, etc), enables the utilization of identifying time-varying bottlenecks. Moreover, Using windows with different

granularity can make it possible to find regularities, dependencies, correlations, seasonality and thus support not only immediate reaction to risky situations, but also other operation activities and planning (e.g. dynamic pipelines with parallel machines schedules on a need basis to alleviate bottlenecks), to reduce overall risks and related costs. For instance, comparing the results of six consecutive one-hour windows with the result of a six-hour window on the same data, can reveal the dependencies of the minor bottlenecks in each one-hour window with the major one in the six-hour window.

Besides other benefits, it is possible to highlight the possibilities of cost-efficiency in data processing, since the aforementioned process can be executed by minimal SPEs, such as e.g. Liebre, that has been shown that it can conduct substantial amount or computation, also on embedded devices, without unnecessary extra costs [11].

B. Data Validation

As discussed in Section I, the data validation round contains three steps; reading data from MES file, sanitizing, and organizing data based on analysis parameters. Figure 4 shows the layout of the proposed validation module by means of the stream processing operators. The module receives flow of MES data as input and produces a flow of sanitized and organized data which is ready to be fed to any analysis application.

To start composing streaming-based queries on MES data, we need to define a sequence of tuples that share a common schema and for that we need to know more about MES data. Each MES data record carries several attributes among which we are intrested in a few one including the creation timestamp, id of the production machine, ANDON lights, and the period during which the data is valid. Table I shows an example MES record of the production line. As shown in the table, each machine has four different ANDON lights which the MES collects and stores. At any instance of time, one or several of lights may be on for each machine. The combination of ANDON lights are used to represent the machine state (active or inactive) as shown in Table II. Using the above mentioned attributes of MES data record, we use the following schema for tuples:

$$< ts, machineID, red, yellow, green, white, duration >$$

where ts is the creation timestamp of the tuple, $machineID$ shows the id of the production machine, red, $yellow$, $green$, and $white$ are ANDON lights, and finally $duration$ indicates the alidity time interval of the data.

timestamp	machineID	duration	ANDON lights			
			r	y	g	w
2015-09-17 12:57:14	M1	273	1	1	0	0
2015-09-17 12:58:20	M2	10	0	0	1	0
2015-09-17 12:58:30	M2	382	0	1	1	0
2015-09-17 13:01:47	M1	85	1	1	0	0

TABLE I: Example MES record of two different machines in a production line

ANDON light	Status	State
yellow	producing	active
green		
white		
yellow+white		
red	down	active
no light		
green+yellow	blocked / starved / idle	inactive
green+white		
green+yellow+white		

TABLE II: Machine states with different combination of ANDON lights being on

In addition to MES data, the validation module requires configuration parameters such as *scheduled hours* and *windows*. Scheduled hours indicate the time during a day that the production line is active (e.g. from 06:00am till 11:59pm on Mondays) and windows show the time granularity within the scheduled hours that the user seeks to identify the bottleneck through (e.g. 6 hours). Figure 5 shows a sample scheduled hours and how the hours of the day are divided into windows. The validation module starts working as soon as it receives the initial values for the configurations. It then keeps working with these configurations as long as there is no update on them. Once user changes the configurations, the module starts working with the new values.

The first step in the validation module is extracting machine data from MES. In order to do so, as shown in Figure 4, a Map operator is used to transform the initial schema of tuples into a new one;

$$< ts, machineID, active, duration >$$

where $active$ is a Boolean attribute that indicates if the machine is active or not. The value of the $active$ attribute is computed using Table II and based on ANDON light attributes presented in the tuple. After transforming the tuples, a Filter operator is used to discard all the tuples that contain `FALSE` as the value of the $active$ attribute.

The second step is to sanitize data by filtering out the noise. In order to detect noise, a Join operator is used to match the MES data with scheduled hours on timestamp attribute and eliminate the tuples that do not match, i.e, are outside the scheduled hours. In a special case, if the production line is active all the time, the scheduled hours will be 24 hours and all the data tuples will be forwarded by the join operator since no tuple is considered as noise. The operator keeps the schema as before.

The third step is to organize data and prepare it for further analysis. For this purpose, we use a Map operator which either forward or splits the tuples. If a tuple lies in more than one window, due to its $duration$ attribute, The Map operator splits the tuple into two or more with the same schema as before but updated ts and $duration$.

The validation module outputs a flow in which tuples lie in only one window. Furthermore, it improves the quality of the data and facilitates the processing procedure which in turn provides the real-time analysis.

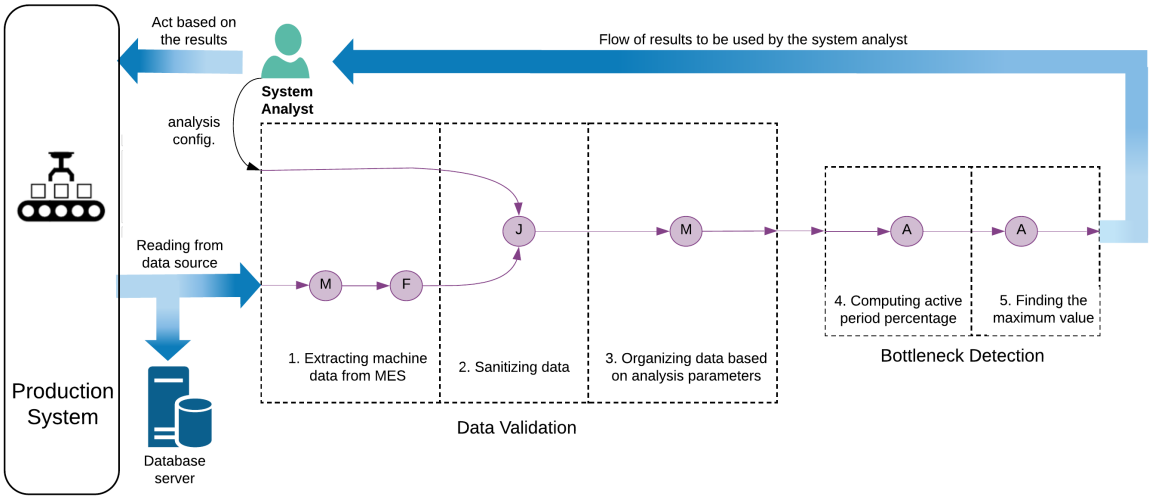

Fig. 4: Layout of the proposed architecture to validate stream of data and detect the bottleneck.

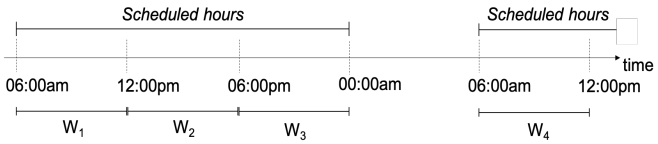

Fig. 5: Example scheduled hours and determined time intervals

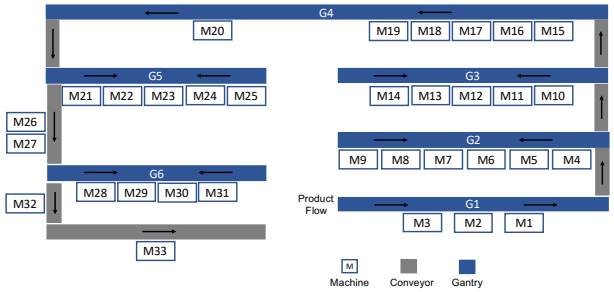

Fig. 6: Layout of the production line

C. Bottleneck Detection Module

The validation module is be followed by an analysis module which receives a flow of sanitized and organized data and continuously perform the bottleneck detection procedure. To perform the bottleneck detection procedure based on the active period percentage, we use an Aggregate operator to group all tuples with the same $machineID$ in the same window and then calculate the active period percentage. The output tuples of the Aggregate operator have a schema composed by attributes

$$< ts, machineID, activePercentage >$$

which shows the active period percentage of a specific machine over a specific window (indicated by ts).

After computing active period percentage for all machines in the window, the final step is to find the machine with the maximum value and report it as the bottleneck of the window. For this purpose, another Aggregate operator is used to find the maximum.

The result of the analysis module is a stream of data in which each tuple indicates the bottlenecks over the user specified window. These results can be used immediately for urgent actions or can be stored for further analysis.

IV. Experimental Study – Evaluation and Discussion

To test and evaluate the performance of the proposed architecture, a real-world MES data from a production line in an automotive manufacturing company in Sweden consisting of parallel machines was taken. Figure 6 shows the studied production system including 33 machines and 6 gantries. The gantries transport the material between the parallel machines. Each machine (as well as the gantries) in the production line is connected to MES in order to monitor the machine status during the production run. The MES data for each machine was collected for the period of two years (from September 2014 to September 2016). The total number of data tuples coming from 39 sources is more than 8.7 million.

The architecture has been implemented in Java on top of Apache Flink, and executed on a 2.10 GHzIntel(R) Xeon(R) E5-2695 system with 38 cores on two sockets (18 cores per socket) and 64 GB memory in total. In our implementation, each of streaming operator is assigned to a processing thread to leverage pipeline parallelism.

In order to identify the bottlenecks on a real-time basis, it is important to reduce the analysis latency. The analysis latency is the timestamp differences of the result tuple for a window and the latest input tuple that produced it. Figure 7 presents the average latency to find the bottleneck over a window for different window sizes. As shown in the figure, regardless of

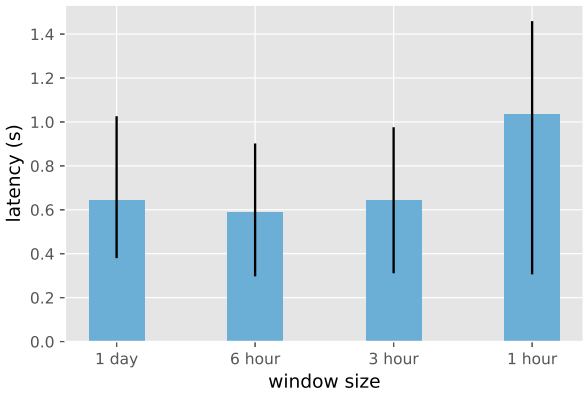

Fig. 7: Average latency of the architecture (including data validation and bottleneck detection) for different window sizes

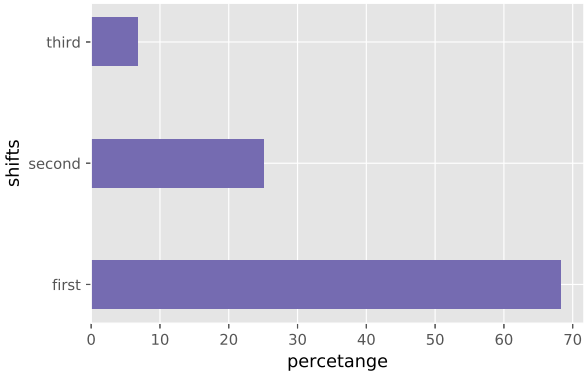

Fig. 8: Number of times that the bottleneck of a day is a bottleneck in any of the shifts during the day

the window size, it takes less than 1 second on average to validate the data and detect the bottleneck, which gives more time to the domain experts operating accurately.

Moreover, in order to study the advantage of having configurable analytics, we run the experiments on the same dataset with two different window sizes; 1-day and 1-shift (one shift is six hours). We then, keep track of the first occurrence of the bottleneck of the day in any of the shifts from the first shift in the morning till the third shift in the evening (Each day includes three shifts). Figure 8 shows the number of times (in percentage) that the bottleneck of the day was also a bottleneck in any of the three shifts. As shown in the figure, 67% of the time, the bottleneck of the day was a bottleneck during the first shift. Considering the results of Figure 7, since it only takes a few milliseconds to process data of one shift (six hours), it is possible to run the analytics during the shifts changeovers and act based on the analysis results in the new shift.

V. Conclusion

We have discussed the stream processing paradigm in the processing of production system data, as an approach that can facilitate automation and provisioning of information useful to detect risks, react to situations and increase effectiveness. Stream processing enables also pipeline parallelism in the data processing, enhancing timeliness, too. Besides illustrating the usage of our methodology on an established method for bottleneck detection, with a large volume of real-system data, we highlighted that the proposed streaming data technique can be coupled with other bottleneck detection algorithm proposed in [7], [10], [14], [15], as well as other problems of continuous monitoring.

References

[1] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, July 2009.

[2] C. Betterton and S. Silver. Detecting bottlenecks in serial production lines–a focus on interdeparture time variance. *International Journal of Production Research*, 50(15):4158–4174, 2012.

[3] J. Bokrantz, A. Skoogh, D. Lämkull, A. Hanna, and T. Perera. Data quality problems in discrete event simulation of manufacturing operations. *Simulation*, 94(11):1009–1025, 2018.

[4] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[5] E. M. Goldratt and J. Cox. *The goal: a process of ongoing improvement*. Routledge, 2016.

[6] V. Gulisano, M. Almgren, and M. Papatriantafilou. Online and scalable data validation in advanced metering infrastructures. In *IEEE PES Innovative Smart Grid Technologies, Europe*, pages 1–6. IEEE, 2014.

[7] W. J. Hopp and M. L. Spearman. Factory physics: Foundations of manufacturing management, burr ridge, il, 2000.

[8] J. Kletti. *Manufacturing Execution System-MES*. Springer, 2007.

[9] D. Laney. 3-d data management: Controlling data volume, velocity and variety. http://blogs.gartner.com. Accessed:2019-4-20.

[10] L. Li, Q. Chang, and J. Ni. Data driven bottleneck detection of manufacturing systems. *International Journal of Production Research*, 47(18):5019–5036, 2009.

[11] H. Najdataei, Y. Nikolakopoulos, V. Gulisano, and M. Papatriantafilou. Continuous and parallel lidar point-cloud clustering. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 671–684. IEEE, 2018.

[12] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou. Genealog: Fine-grained data streaming provenance at the edge. In *Proceedings of the 19th International Middleware Conference*, pages 227–238. ACM, 2018.

[13] C. Roser, M. Nakano, and M. Tanaka. A practical bottleneck detection method. In *Proceedings of the 33nd conference on Winter simulation*, pages 949–953. IEEE Computer Society, 2001.

[14] S. Sengupta, K. Das, and R. P. Vantil. A new method for bottleneck detection. In *Simulation Conference, 2008. WSC 2008. Winter*, pages 1741–1745. IEEE, 2008.

[15] M. Subramaniyan, A. Skoogh, H. Salomonsson, P. Bangalore, M. Gopalakrishnan, and A. Sheikh Muhammad. Data-driven algorithm for throughput bottleneck analysis of production systems. *Production & Manufacturing Research*, 6(1):225–246, 2018.

[16] J. van Rooij, J. Swetzén, V. Gulisano, M. Almgren, and M. Papatriantafilou. echidna: Continuous data validation in advanced metering infrastructures. In *2018 IEEE International Energy Conference (ENERGYCON)*, pages 1–6. IEEE, 2018.

[17] C. Yu and A. Matta. Data-driven bottleneck detection in manufacturing systems: A statistical approach. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 710–715. IEEE, 2014.

[18] J. Zou, Q. Chang, J. Huang, J. Arinez, and G. Xiao. Analysis of end-of-state impact on manufacturing system production performance. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 823–828. IEEE, 2018.